Manual Técnico del Sistema de Registro Vehicular

1. Introducción

Este documento describe los aspectos técnicos del sistema de Registro Vehicular desarrollado en Java utilizando NetBeans. El sistema permite la gestión descentralizada de vehículos, traspasos y multas por departamento, con soporte para estructuras de datos como árboles binarios de búsqueda (ABB), árboles AVL, listas dobles y circulares.

2. Arquitectura del Sistema

2.1 Arquitectura General

El sistema sigue una arquitectura modular basada en el patrón Modelo-Vista-Controlador (MVC):

- **Modelo:** Maneja los datos (vehículos, traspasos, multas) y estructuras lógicas (ABB, AVL, listas).
- **Vista:** Interfaz gráfica desarrollada con Java Swing.
- **Controlador:** Orquesta la lógica entre vista y modelo, controla eventos y actualiza vistas.

2.2 Módulos Principales

- **Busqueda:** Selección de carpeta base por departamento.
- **Interfaz:** Menú principal con acceso a distintos módulos.
- **Vehiculos, Traspasos, Multas:** Interfaces separadas para cada entidad.
- **Estadisticas:** Consulta de datos agregados y comparación de estructuras.
- **Graficador:** Visualización de árboles usando Graphviz.

3. Estructuras de Datos Utilizadas

3.1 Árbol Binario de Búsqueda (ABB)

- Usado para almacenar y buscar datos de vehículos y multas por placa.
- Permite inserción, eliminación y recorrido inOrden para visualización ordenada. **3.2 Árbol AVL**

- Variante auto-balanceada del ABB.
- Mejora el tiempo de búsqueda con equilibrio automático mediante rotaciones.
- Usado paralelamente al ABB para comparar eficiencia.

3.3 Listas Doblemente Enlazadas

- Manejo de traspasos por vehículo.
- Permite recorrido en ambas direcciones para ver historial completo.

3.4 Listas Circulares

- Usadas en funciones de control rotativo o cola de espera en versiones futuras.

3.5 Arreglos

- Se emplean para representar tablas en la interfaz (`JTable`) y para consolidar datos temporalmente.

4. Algoritmos Implementados

4.1 Inserción en ABB

```
if (nuevo.getPlaca().compareTo(actual.getPlaca()) < 0) {
    actual.setIzquierda(insertar(actual.getIzquierda(), nuevo));
} else {
    actual.setDerecha(insertar(actual.getDerecha(), nuevo));
}
```

4.2 Rotaciones en AVL

- **Rotación Simple a la Derecha:**

```
NodoAVL nuevaRaiz = nodo.izquierda;
nodo.izquierda = nuevaRaiz.derecha;
nuevaRaiz.derecha = nodo;
actualizarAltura(nodo);
actualizarAltura(nuevaRaiz);
return nuevaRaiz;
```

4.3 Búsqueda por placa en ABB/AVL

```
if (placa.equals(nodo.getPlaca())) return nodo;
else if (placa.compareTo(nodo.getPlaca()) < 0) return buscar(nodo.izquierda,
placa);
else return buscar(nodo.derecha, placa);
```

4.4 Recorrido InOrden para llenar JTable

```
void inOrden(Nodo actual) {
    if (actual != null) {
        inOrden(actual.izquierda);
        modeloTabla.addRow(new Object[]{...});
        inOrden(actual.derecha);
    }
}
```

4.5 Carga de archivos

```
File[] archivos = carpeta.listFiles();
for (File archivo : archivos) {
    if (archivo.getName().endsWith(".txt")) {
        Scanner lector = new Scanner(archivo);
        ...
    }
}
```

5. Observaciones Finales

- La integración de estructuras ABB y AVL permite comparar el rendimiento real en operaciones de inserción y búsqueda.
- La modularidad del sistema facilita su mantenimiento y extensión.
- La arquitectura MVC permite mantener una separación clara entre lógica, datos y presentación.

Versión del sistema: 24.0\ **Lenguaje:** Java\ **Entorno de desarrollo:** NetBeans\ **Autor:** Nincy Abigail Rodriguez Chavez